计算机学院专业必修课

计算机组成

Cache

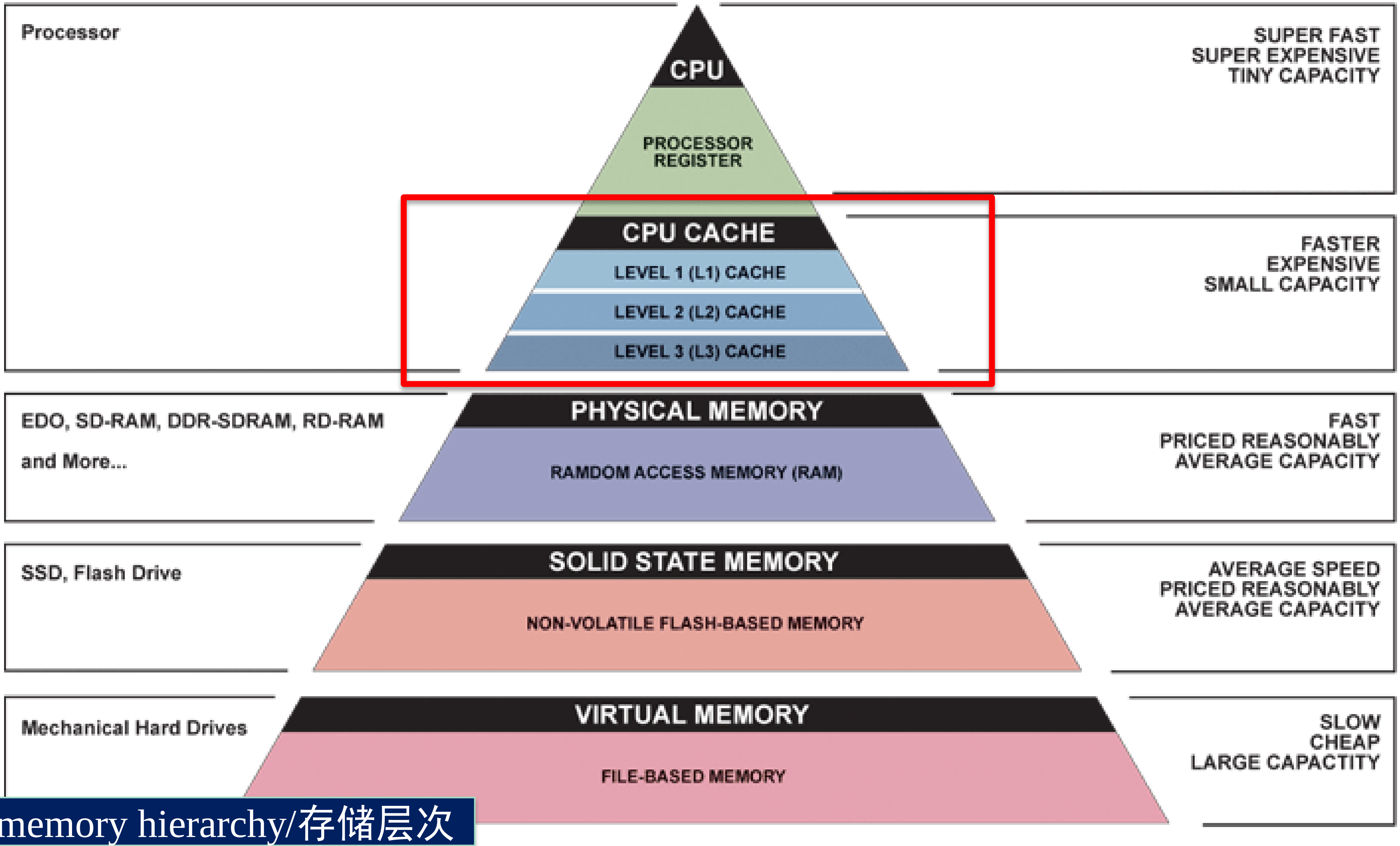
高小鹏

北京航空航天大学计算机学院

提纲

- 内容主要取材：CS61C的11讲和12讲
- 存储层次概述
- 直接映射Cache
- 直接映射Cache举例
- Cache读和写
- Cache性能
- 多级Cache
- 组相连Cache
- 改进Cache性能
- 多级Cache性能实战
- 当代Cache举例

Great Idea #3: Principle of Locality/ Memory Hierarchy



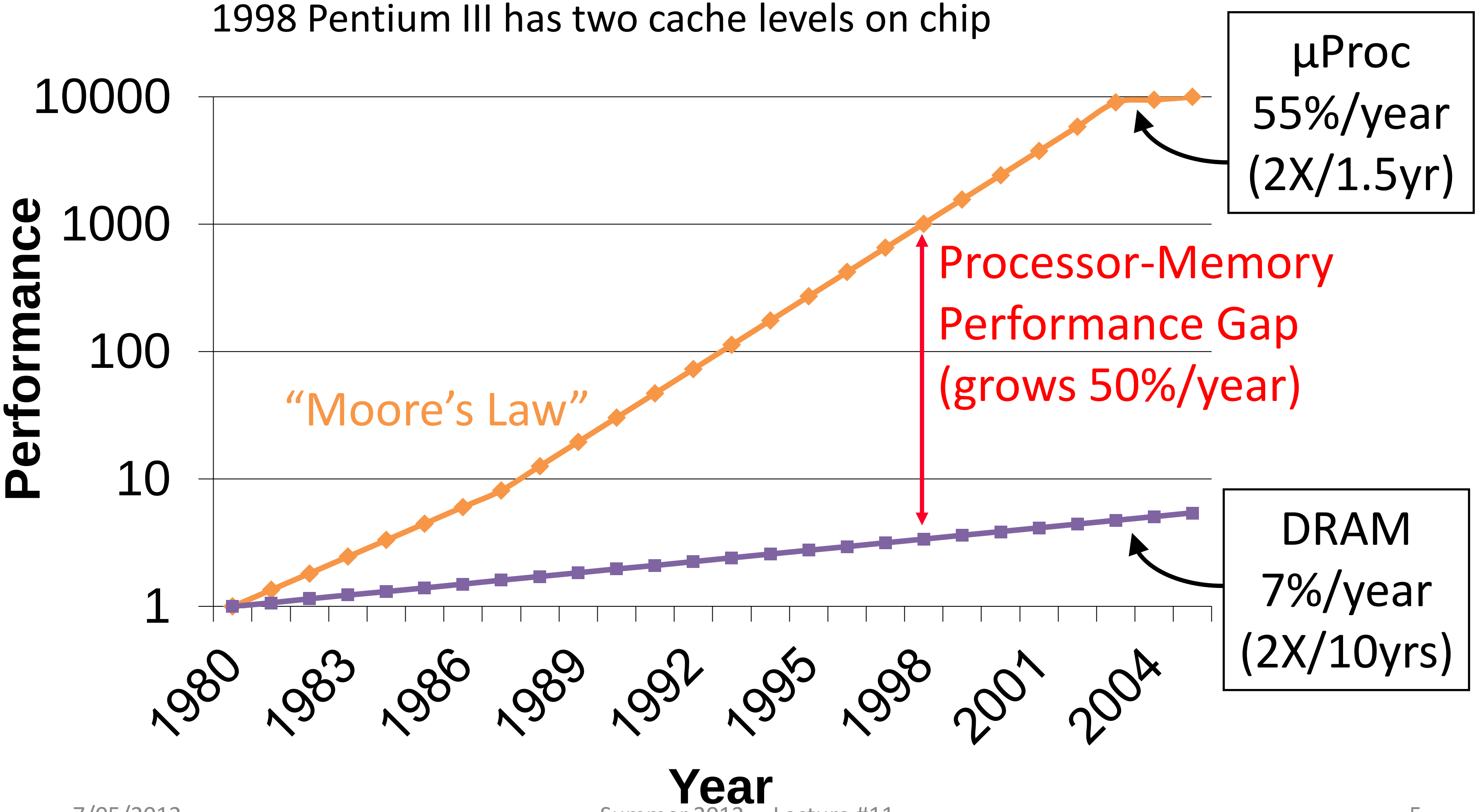
memory hierarchy/存储层次

Storage in a Computer

- Processor
  - – Holds data in register files (~ 100 bytes)
  - – Registers accessed on sub-nanosecond timescale
- Memory ("main memory")
  - – More capacity than registers (~ Gbytes)
  - – Access time ~ 50-100 ns
- Hundreds of clock cycles per memory access?!

Processor-Memory Gap

1989 first Intel CPU with cache on chip
1998 Pentium III has two cache levels on chip



µProc
55%/year
(2X/1.5yr)

"Moore's Law"

Processor-Memory
Performance Gap
(grows 50%/year)

DRAM
7%/year
(2X/10yrs)

Principle of Locality (1/3)

- *Principle of Locality:* Programs access only a small portion of the full address space at any instant of time
  - **Recall:** Address space holds both code and data
  - Loops and sequential instruction execution mean generally localized code access
  - Stack and Heap try to keep your data together
  - Arrays and structs naturally group data you would access together

locality/局部性

Principle of Locality (2/3)

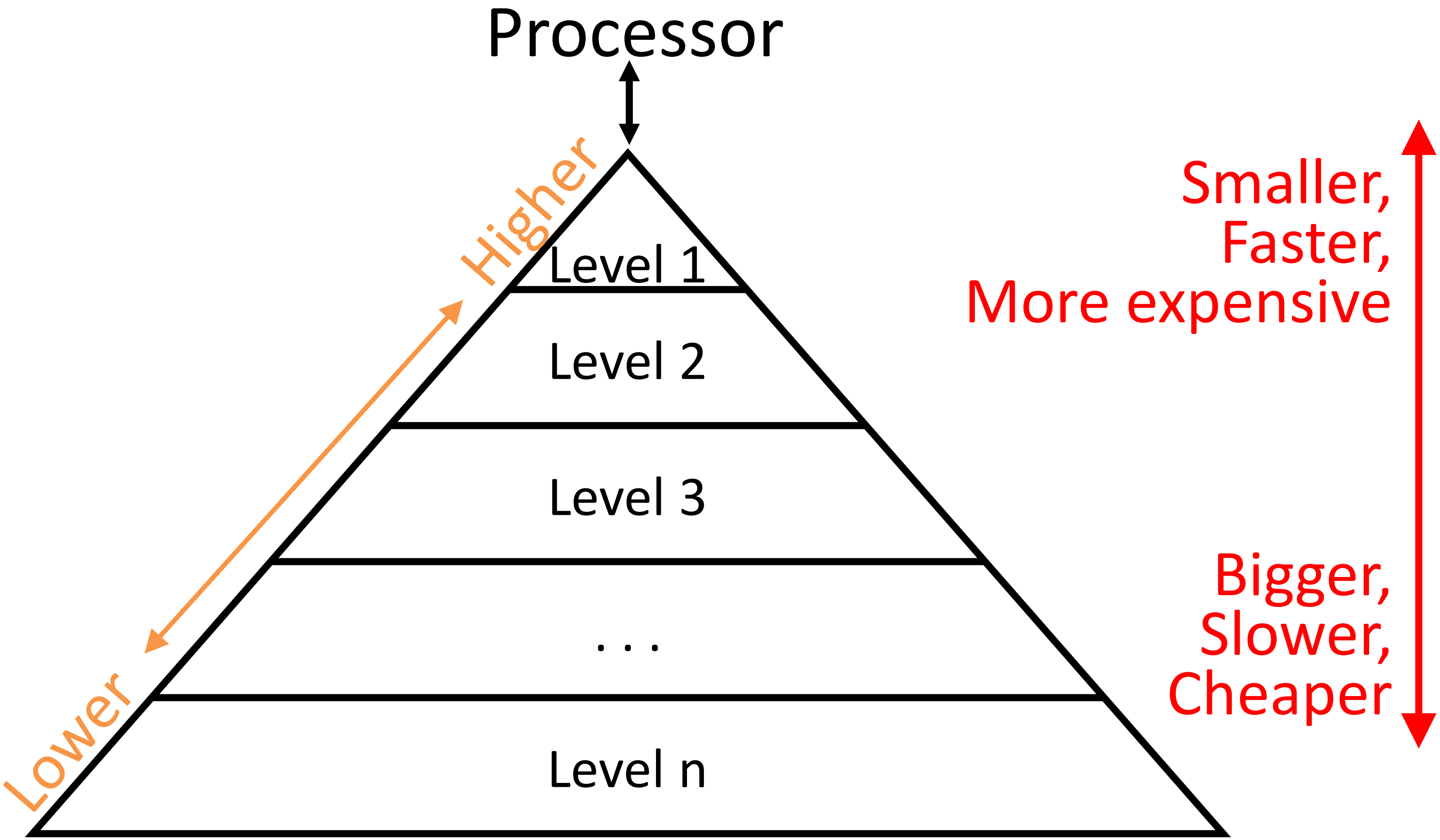
- *Temporal Locality* (locality in time)
  - ~~Go back to the same book on desk multiple times~~
  - If a memory location is referenced then it will tend to be referenced again soon
- *Spatial Locality* (locality in space)
  - ~~When go to book shelf, grab many books on J.D. Salinger since library stores related books together~~
  - If a memory location is referenced, the locations with nearby addresses will tend to be referenced soon

temporal locality/时间局部性 spatial locality/空间局部性

Principle of Locality (3/3)

- We exploit the principle of locality in hardware via a *memory hierarchy* where:
  - Levels closer to processor are faster (and more expensive per bit so smaller)
  - Levels farther from processor are larger (and less expensive per bit so slower)
- **Goal:** Create the illusion of memory being almost as fast as fastest memory and almost as large as biggest memory of the hierarchy

Memory Hierarchy Schematic



Cache Concept

- Introduce intermediate hierarchy level: memory *cache*, which holds a copy of a subset of main memory
  - As a pun, often use $ ("cash") to abbreviate cache (e.g. D$ = Data Cache, L1$ = Level 1 Cache)
- Modern processors have separate caches for instructions and data, as well as several levels of caches implemented in different sizes
- Implemented with same IC processing technology as CPU and integrated on-chip – faster but more expensive than main memory

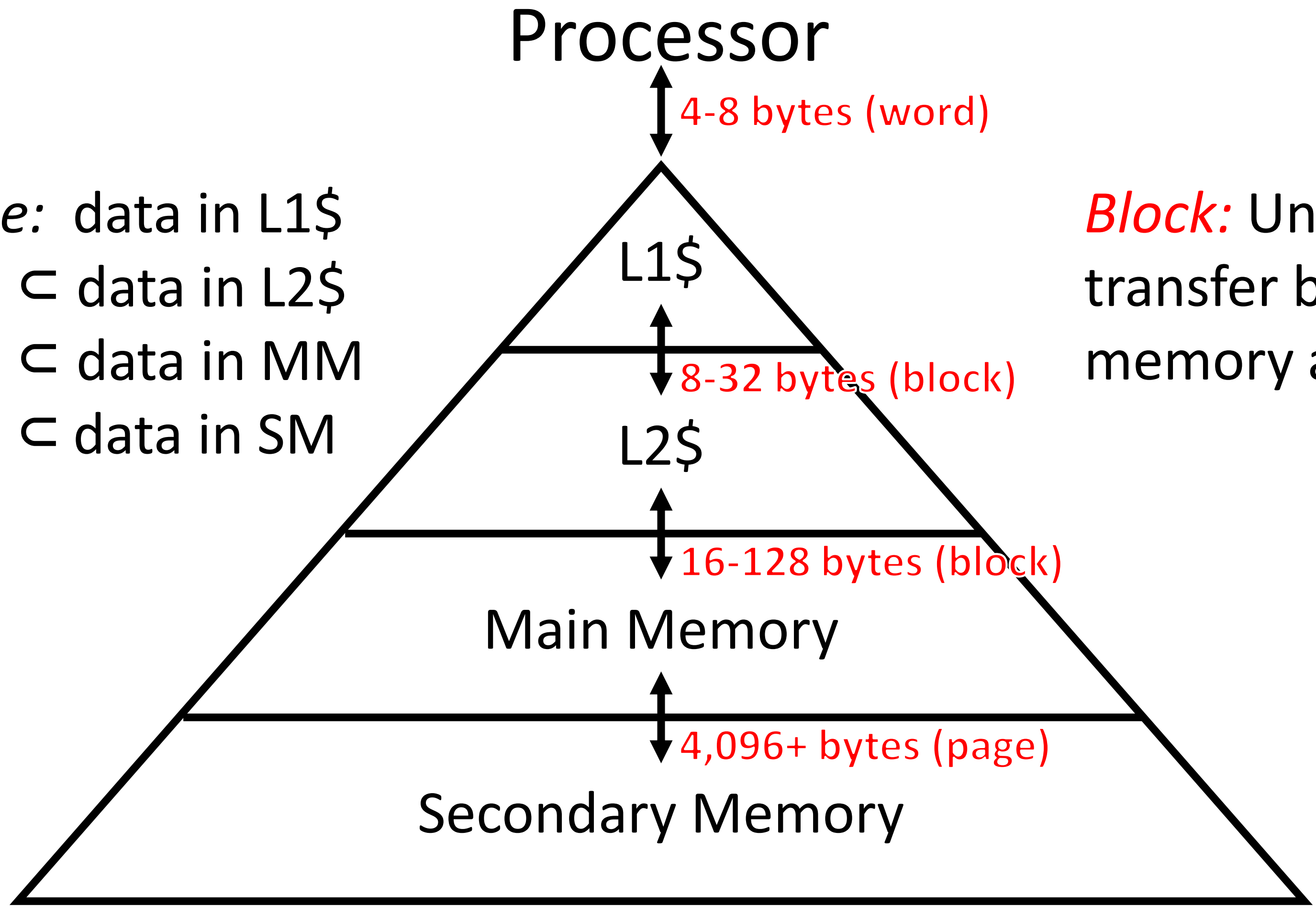
Memory Hierarchy Technologies

- Caches use static RAM (SRAM)
  - + Fast (typical access times of 0.5 to 2.5 ns)
  - – Low density (6 transistor cells), higher power, expensive ($2000 to $4000 per GB in 2011)
  - *Static:* content will last as long as power is on

- Main memory uses dynamic RAM (DRAM)
  - + High density (1 transistor cells), lower power, cheaper ($20 to $40 per GB in 2011)
  - – Slower (typical access times of 50 to 70 ns)
  - *Dynamic:* needs to be "refreshed" regularly (~ every 8 ms)

Memory Transfer in the Hierarchy

Inclusive: data in L1$
⊂ data in L2$
⊂ data in MM
⊂ data in SM



Block: Unit of transfer between memory and cache

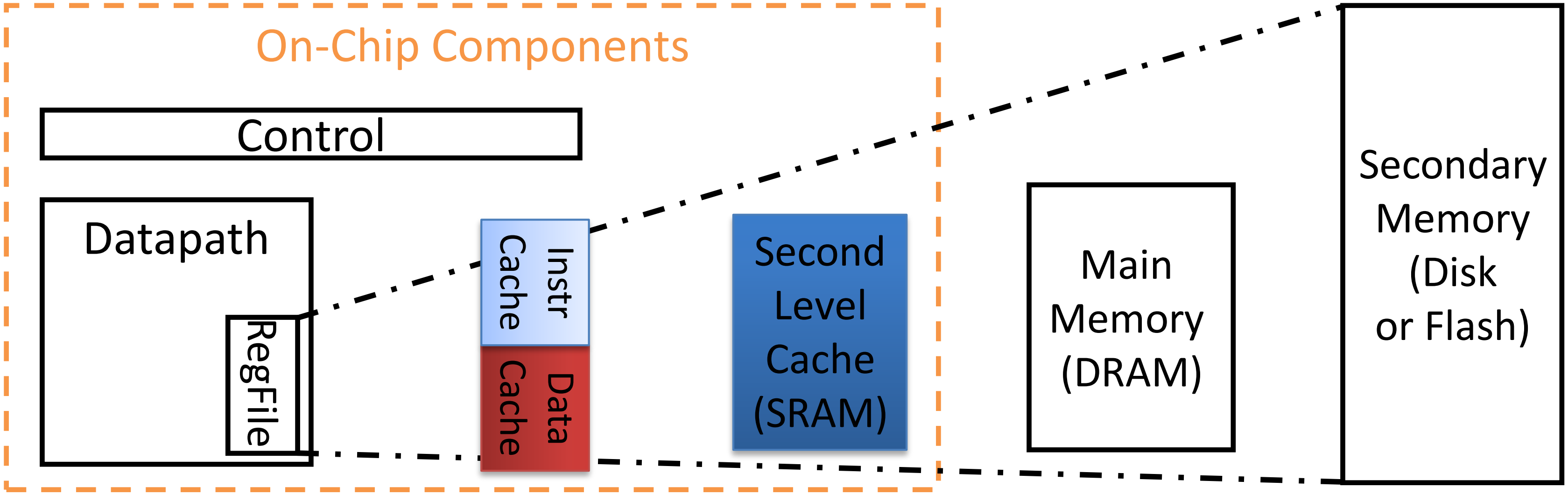
Managing the Hierarchy

- registers ↔ memory
  - By compiler (or assembly level programmer)
- cache ↔ main memory
  - By the cache controller hardware

We are here

- main memory ↔ disks (secondary storage)
  - By the OS (virtual memory, which is a later topic)
  - Virtual to physical address mapping assisted by the hardware (TLB)
  - By the programmer (files)

Typical Memory Hierarchy



Speed: (cycles)	½'s	1's	10's	100's	1,000,000's
Size: (bytes)	100's	10K's	M's	G's	T's
Cost/bit:	highest ⟷				lowest

提纲

- 内容主要取材
  - CS61C的11讲和12讲
- 存储层次概述
- 直接映射Cache
- 直接映射Cache举例
- Cache读和写
- Cache性能
- 多级Cache
- 组相连Cache
- 改进Cache性能
- 多级Cache性能实战
- 当代Cache举例

Cache Management

- ~~Library analogy: organization is necessary!~~

- What is the overall organization of blocks we impose on our cache?
  - Where do we put a block of data from memory?
  - How do we know if a block is already in cache?
  - How do we quickly find a block when we need it?
  - When do we replace something in the cache?

General Notes on Caches

- **Recall:** Memory is *byte-addressed*
- We haven't specified the size of our "blocks," but will be multiple of word size (32-bits)
  - How do we access individual words or bytes within a block? OFFSET
- Cache is smaller than memory
  - Can't fit all blocks at once, so multiple blocks in memory map to the same cache slot (*row*) INDEX
  - Need some way of identifying which memory block is currently in the row TAG

Direct-Mapped Caches (1/3)

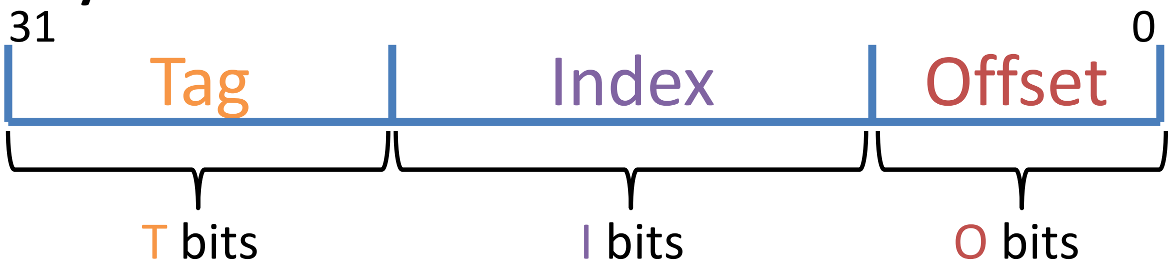
- Each memory block is mapped to *exactly one row* in the cache (*direct-mapped*)
  - Use simple hash function
- Effect of block size:
  - Spatial locality dictates our blocks consist of adjacent bytes, which differ in address by 1
  - Offset field: Lowest bits of memory address can be used to index to specific bytes within a block
    - Block size needs to be a power of two (in bytes)

Direct-Mapped Caches (2/3)

- Effect of cache size: (total stored data)
  - Determines number of blocks cache holds
  - ~~If could hold all of memory, would use remaining bits (minus offset bits) to select appropriate row of cache~~
  - Index field: Apply hash function to remaining bits to determine *which row* the block goes in
    - (*block* address) modulo (# of *blocks* in the cache)
  - Tag field: Leftover upper bits of memory address determine *which portion of memory* the block came from (identifier)

TIO Address Breakdown

- Memory address fields:



31 0

Tag Index Offset

T bits I bits O bits

- Meaning of the field sizes:
  - O bits ↔ 2^{O} bytes/block = 2^{O-2} words/block
  - I bits ↔ 2^{I} rows in cache = cache size / block size
  - T bits = A – I – O, where A = # of address bits (A = 32 here)

Direct-Mapped Caches (3/3)

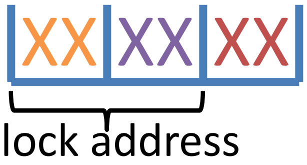
- What's actually in the cache?
  - Each row contains the actual data block to store (B bits = 8 × 2^{O} bits)
  - In addition, must save Tag field of address as identifier (T bits)
  - Valid bit: Indicates whether the block in that row is valid or not
- Total bits in cache = # rows × (B + T + 1)

  $= 2^{I} \times (8 \times 2^{O} + T + 1)$ bits

Cache Example (1/2)

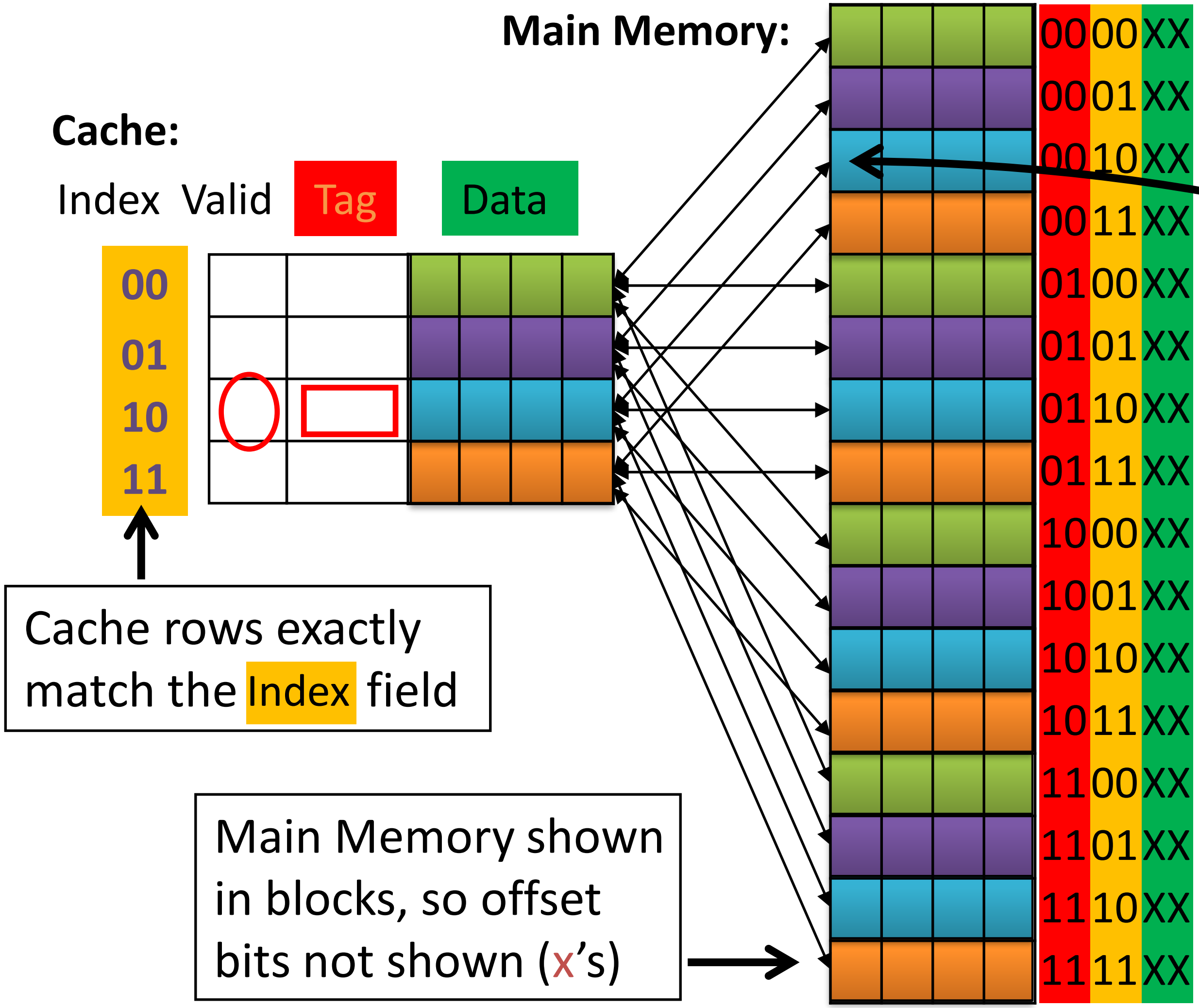
- Cache parameters:
  - Address space of 64B, block size of 1 word, cache size of 4 words
- TIO Breakdown:

Memory Addresses:



  - 1 word = 4 bytes, so O = $\log_2(4) = 2$
  - Cache size / block size = 4, so I = $\log_2(4) = 2$
  - A = $\log_2(64) = 6$ bits, so T = $6 - 2 - 2 = 2$
- Bits in cache = $2^2 \times (8\times2^2 + 2 + 1) = 140$ bits

Cache Example (2/2)



Cache rows exactly match the Index field

Main Memory shown in blocks, so offset bits not shown (x's)

Which blocks map to each row of the cache? (see colors)

On a memory request: (let's say 001011_{two})

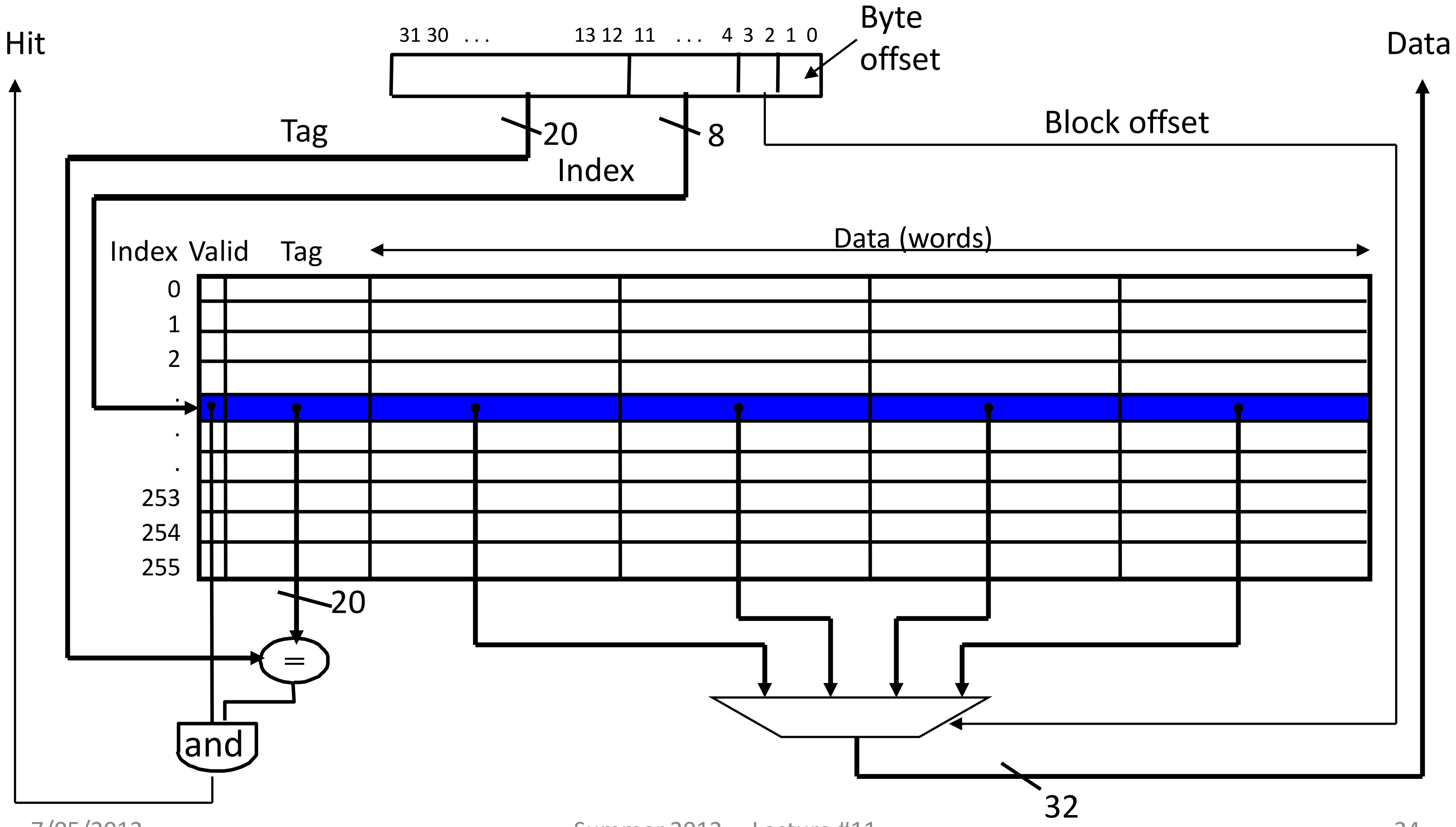
1) Take Index field 10

2) Check if Valid bit is true in that row of cache

3) If valid, then check if Tag matches

Direct-Mapped Cache Internals

- 4 words/block, cache size = 1 Ki words



Caching Terminology (1/2)

- When reading memory, 3 things can happen:
  - Cache hit:
    Cache block is valid and contains the proper address, so read the desired word
  - Cache miss:
    Nothing in that row of the cache (not valid), so fetch from memory
  - Cache miss with block replacement:
    Wrong block is in the row, so discard it and fetch desired data from memory

terminology/术语 miss/缺失

Caching Terminology (2/2)

- How effective is your cache?
  - Want to max cache hits and min cache misses
  - Hit rate (HR): Percentage of memory accesses in a program or set of instructions that result in a cache hit
  - Miss rate (MR): Like hit rate, but for cache misses
    MR = 1 – HR
- How fast is your cache?
  - Hit time (HT): Time to access cache (including Tag comparison)
  - Miss penalty (MP): Time to replace a block in the cache from a lower level in the memory hierarchy

penalty/代价

Sources of Cache Misses: The 3Cs

- Compulsory: (cold start or process migration, 1st reference)
  - First access to block impossible to avoid; Effect is small for long running programs
- Capacity:
  - Cache cannot contain all blocks accessed by the program
- Conflict: (collision)
  - Multiple memory locations mapped to the same cache location

compulsory/强制

提纲

- 内容主要取材
  - CS61C的11讲和12讲
- 存储层次概述
- 直接映射Cache
- 直接映射Cache举例
- Cache读和写
- Cache性能
- 多级Cache
- 组相连Cache
- 改进Cache性能
- 多级Cache性能实战
- 当代Cache举例

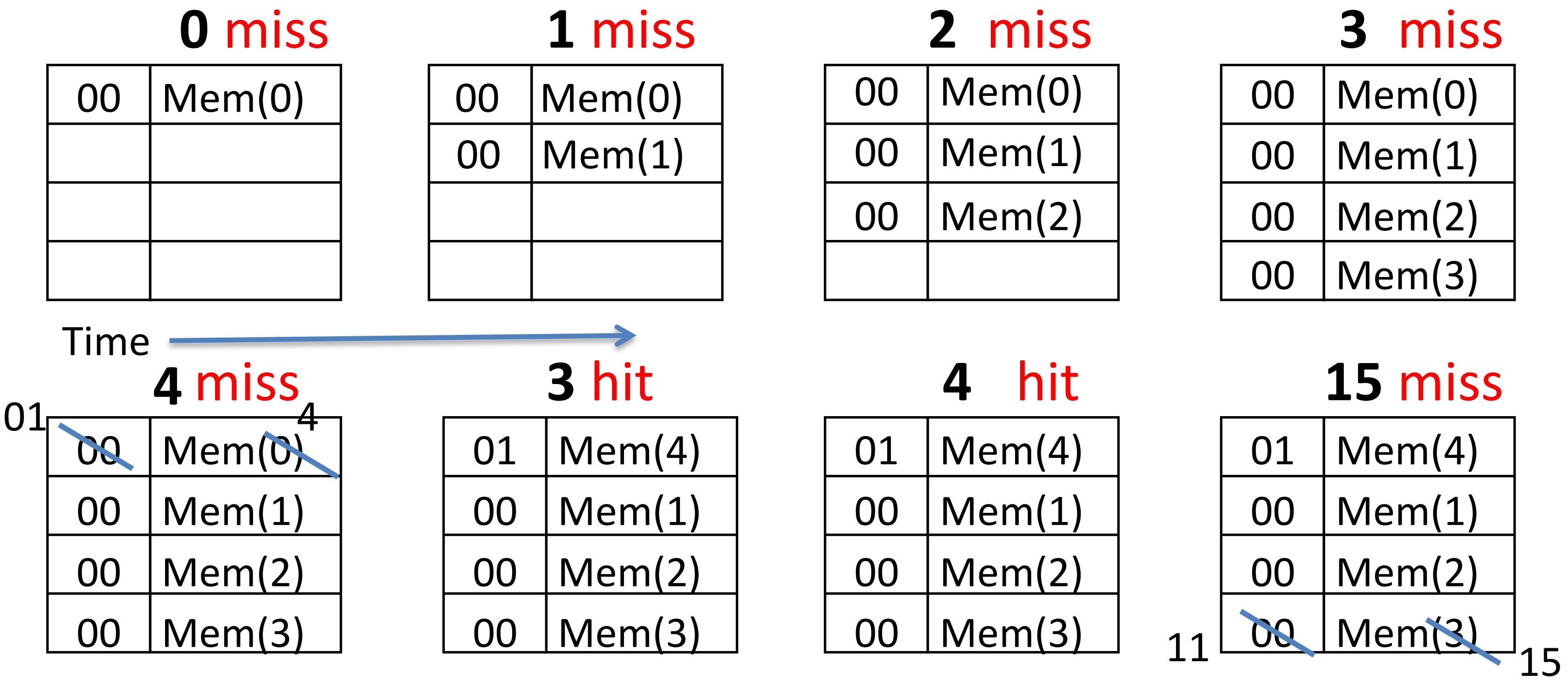
Direct-Mapped Cache Example

(modified by GXP)

- Consider the sequence of memory address accesses

Start with an empty cache - all blocks initially marked as not valid

0	1	2	3	4	3	4	15
0000	0001	0010	0011	0100	0011	0100	1111



0 miss

00	Mem(0)

1 miss

00	Mem(0)
00	Mem(1)

2 miss

00	Mem(0)
00	Mem(1)
00	Mem(2)

3 miss

00	Mem(0)
00	Mem(1)
00	Mem(2)
00	Mem(3)

Time →

4 miss

01 ~~00~~	~~Mem(0)~~ 4
00	Mem(1)
00	Mem(2)
00	Mem(3)

3 hit

01	Mem(4)
00	Mem(1)
00	Mem(2)
00	Mem(3)

4 hit

01	Mem(4)
00	Mem(1)
00	Mem(2)
00	Mem(3)

15 miss

01	Mem(4)
00	Mem(1)
00	Mem(2)
11 ~~00~~	~~Mem(3)~~ 15

- 8 requests, 6 misses (HR = 0.25, MR = 0.75)

地址空间：16B，block size：1B，cache size：4B
TIO = 2-2-0

Taking Advantage of Spati

- Let cache block hold more than one by

TIP：
不能仅仅通过少数案例来判断cache结构的优劣

Start with an empty cache - all blocks initially marked as not valid

0	1	2	3				
0000	0001	0010	0011	0100	0011	0100	1111

0 miss

00	Mem(1)	Mem(0)

1 hit

00	Mem(1)	Mem(0)

2 miss

00	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

3 hit

00	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

4 miss

01 ~~00~~	~~Mem(1)~~ 5	~~Mem(0)~~ 4
00	Mem(3)	Mem(2)

3hit

01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

4 hit

01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

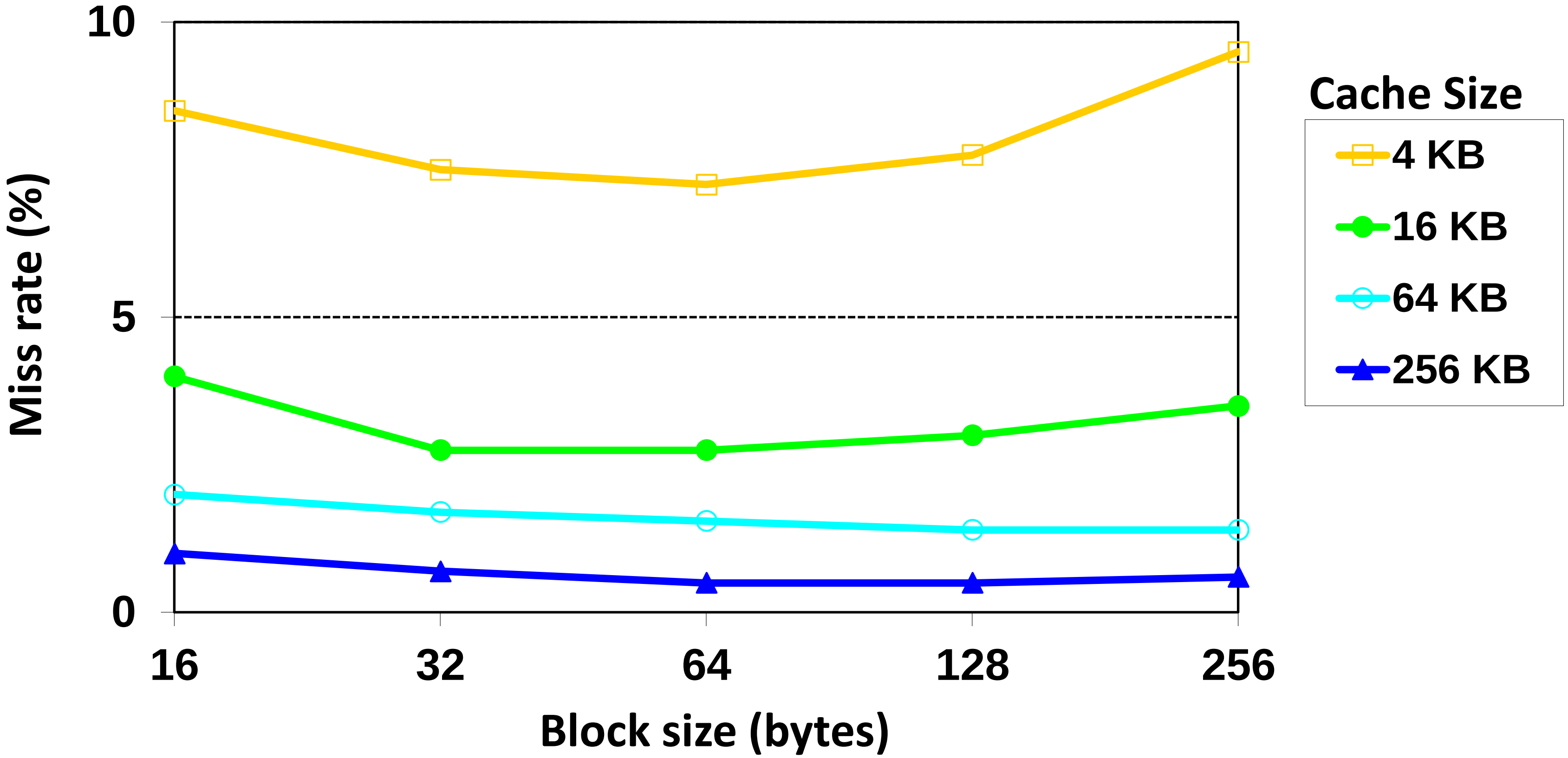
15miss

01	Mem(5)	Mem(4)
11 ~~00~~	~~Mem(3)~~ 15	~~Mem(2)~~ 14

- 8 requests, 4 misses（HR = 0.5， MR = 0.5）

地址空间：16B，block size：2B，cache size：4B
TIO = 2-1-1

Effect of Block and Cache Sizes on Miss Rate



- Miss rate goes up if the block size becomes a significant fraction of the cache size because the number of blocks that can be held in the same size cache is smaller (increasing capacity misses)

提纲

- 内容主要取材
  - CS61C的11讲和12讲
- 存储层次概述
- 直接映射Cache
- 直接映射Cache举例
- Cache读和写
- Cache性能
- 多级Cache
- 组相连Cache
- 改进Cache性能
- 多级Cache性能实战
- 当代Cache举例

Cache一致性问题

时间	事件	Cache内容	位置X的主存内存
0			1
1	CPU读X	1	1
2	CPU将0写入X	0	1

Cache Reads and Writes

- Want to handle reads and writes quickly while maintaining consistency between cache and memory (i.e. both know about all updates)
  - Policies for cache hits and misses are independent
- Here we assume the use of separate instruction and data caches (I$ and D$)
  - Read from both
  - Write only to D$ (assume no self-modifying code)

consistency/一致性

Handling Cache Hits

- Read hits (I$ and D$)
  - Fastest possible scenario, so want more of these
- Write hits (D$)
  1) Write-Through Policy: Always write data to cache and to memory (*through* cache)
     - Forces cache and memory to always be consistent
     - Slow! (every memory access is long)
     - Include a *Write Buffer* that updates memory in parallel with processor

Assume present in all schemes when writing to memory

Handling Cache Hits

- Read hits (I$ and D$)
  - – Fastest possible scenario, so want more of these
- Write hits (D$)
  2) Write-Back Policy: Write data only to cache, then update memory when block is removed
     - Allows cache and memory to be inconsistent
     - Multiple writes collected in cache; single write to memory per block
     - Dirty bit: Extra bit per cache row that is set if block was written to (is "dirty") and needs to be written back

处理Cache缺失

- 缺失代价：随着block变大而变大
- 读缺失（I$和D$）
  - ①流水线暂停；②从主存读入block；③写入cache；④发送word至流水线；⑤流水线恢复执行
- 写缺失（D$）
  - 写分配（write allocate）：①流水线暂停；②从主存读入block；③写入cache；④执行写命中
    - 无论写通还是写回，写分配均有效
    - 可以确保在写缺失后，cache有最新数据

流水线寄存器的Verilog代码示例

- 缺失代价：随着block变大而变大
- 读缺失（I$和D$）
  - ①流水线暂停；②从主存读入block；③写入cache；④发送word至流水线；⑤流水线恢复执行
- 写缺失（D$）
  - 非写分配（No-write allocate）：不写cache而直接写主存
    - cache不包含最新数据
    - 但主存包含最新数据

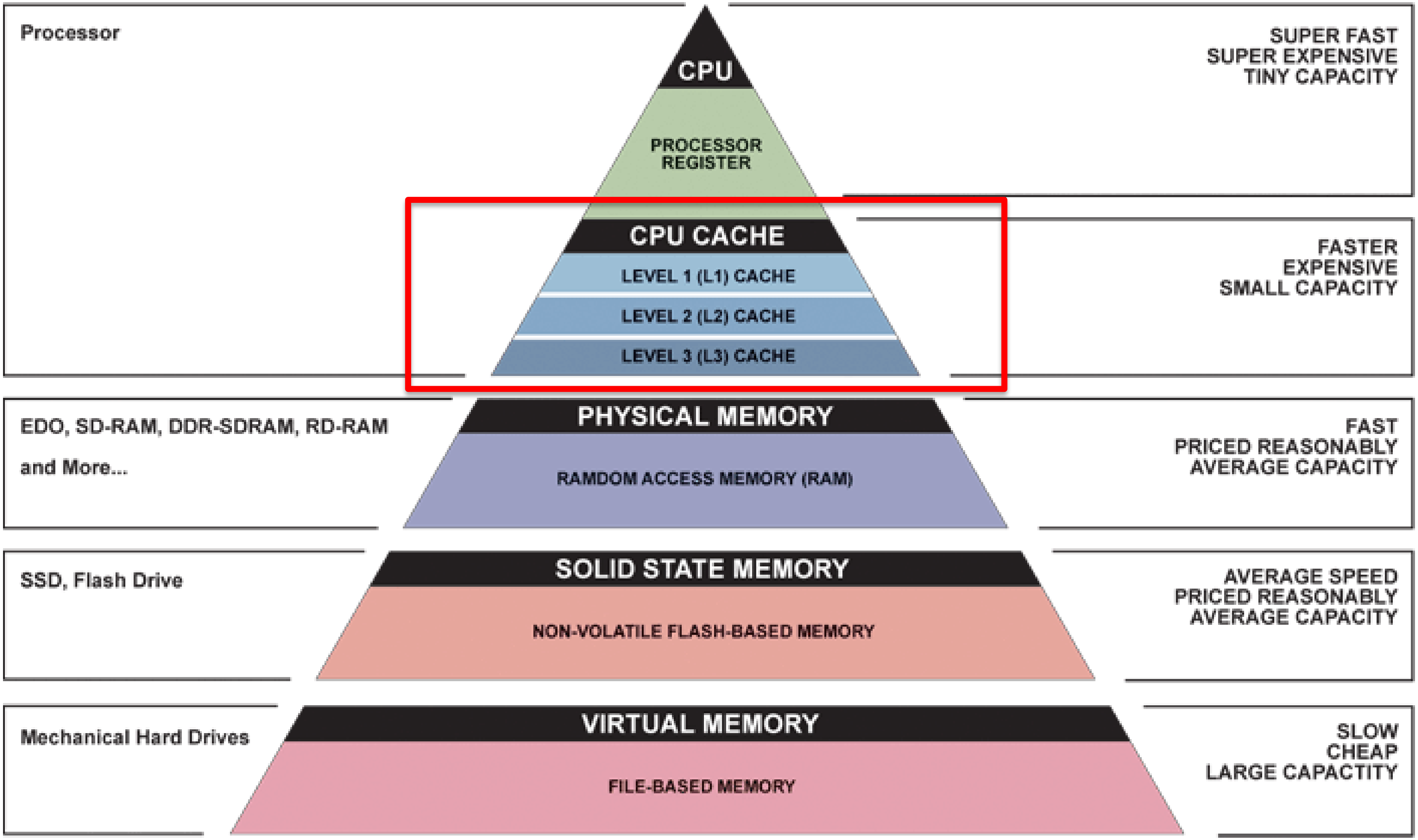
Summary

- Memory hierarchy exploits principle of locality to deliver lots of memory at fast speeds
- Direct-Mapped Cache: Each block in memory maps to exactly one row in the cache
  - Index to determine which row
  - Offset to determine which byte within block
  - Tag to identify if it's the block you want
- Cache read and write policies:
  - *Write-back* and *write-through* for hits
  - *Write allocate* and *no-write allocate* for misses

提纲

- 内容主要取材
  - CS61C的11讲和12讲
- 存储层次概述
- 直接映射Cache
- 直接映射Cache举例
- Cache读和写
- Cache性能
- 多级Cache
- 组相连Cache
- 改进Cache性能
- 多级Cache性能实战
- 当代Cache举例

Great Idea #3: Principle of Locality/ Memory Hierarchy



Cache Performance

- Two things hurt the performance of a cache:
  - Miss rate and miss penalty
- *Average Memory Access Time* (AMAT): average time to access memory considering both hits and misses

AMAT = Hit time + Miss rate × Miss penalty

(abbreviated AMAT = HT + MR × MP)

AMAT Example Usage

- **Processor specs:** 200 ps clock, MP of 50 clock cycles, MR of 0.02 misses/instruction, and HT of 1 clock cycle

  AMAT = 1 + 0.02 × 50 = 2 clock cycles = 400 ps

- Which improvement would be best?
  - 190 ps clock 380 ps
  - MP of 40 clock cycles 360 ps
  - MR of 0.015 misses/instruction 350 ps

Cache Parameter Example

- What is the potential impact of much larger cache on AMAT? (same block size)
  1) Increase HR
  2) Longer HT: smaller is faster
  - At some point, increase in hit time for a larger cache may overcome the improvement in hit rate, yielding a decrease in performance
- Effect on TIO? Bits in cache? Cost?

Effect of Cache Performance on CPI

- **Recall:** CPU Performance

$$\text{CPU Time} = \underset{\text{(IC)}}{\text{Instructions}} \times \text{CPI} \times \underset{\text{(CC)}}{\text{Clock Cycle Time}}$$

- Include memory accesses in CPI:

$$\text{CPI}_{\text{stall}} = \text{CPI}_{\text{base}} + \text{Average Memory-stall Cycles}$$

$$\text{CPU Time} = \text{IC} \times \text{CPI}_{\text{stall}} \times \text{CC}$$

- Simplified model for memory-stall cycles:

$$\text{Memory-stall cycles} = \frac{\text{Accesses}}{\text{Instruction}} \times \text{MR} \times \text{MP}$$

  – We will discuss more complicated models soon

CPI Example

- **Processor specs:** CPI_{base} of 1, a 100 cycle MP, 36% load/store instructions, and 2% I$ and 4% D$ MRs
  - How many times per instruction do we access the I$? The D$?
  - MP is assumed the same for both I$ and D$
  - Memory-stall cycles will be sum of stall cycles for both I$ and D$

CPI Example

- **Processor specs:** CPI$_{base}$ of 1, a 100 cycle MP, 36% load/store instructions, and 2% I$ and 4% D$ MRs
  - Memory-stall cycles

$$= (\underbrace{100\% \times 2\%}_{\text{I\$}} + \underbrace{36\% \times 4\%}_{\text{D\$}}) \times 100 = 3.44$$

  - CPI$_{stall}$ = 1 + 3.44 = 4.44 (more than 3x CPI$_{base}$!)

- ~~What if the CPI$_{base}$ is reduced to 1?~~
- What if the D$ miss rate went up by 1%?

The 3Cs Revisited: Design Solutions

- Compulsory:
  - Increase block size (increases MP; too large blocks could increase MR)
- Capacity:
  - Increase cache size (may increase HT)
- Conflict:
  - Increase cache size
  - Increase associativity (may increase HT)

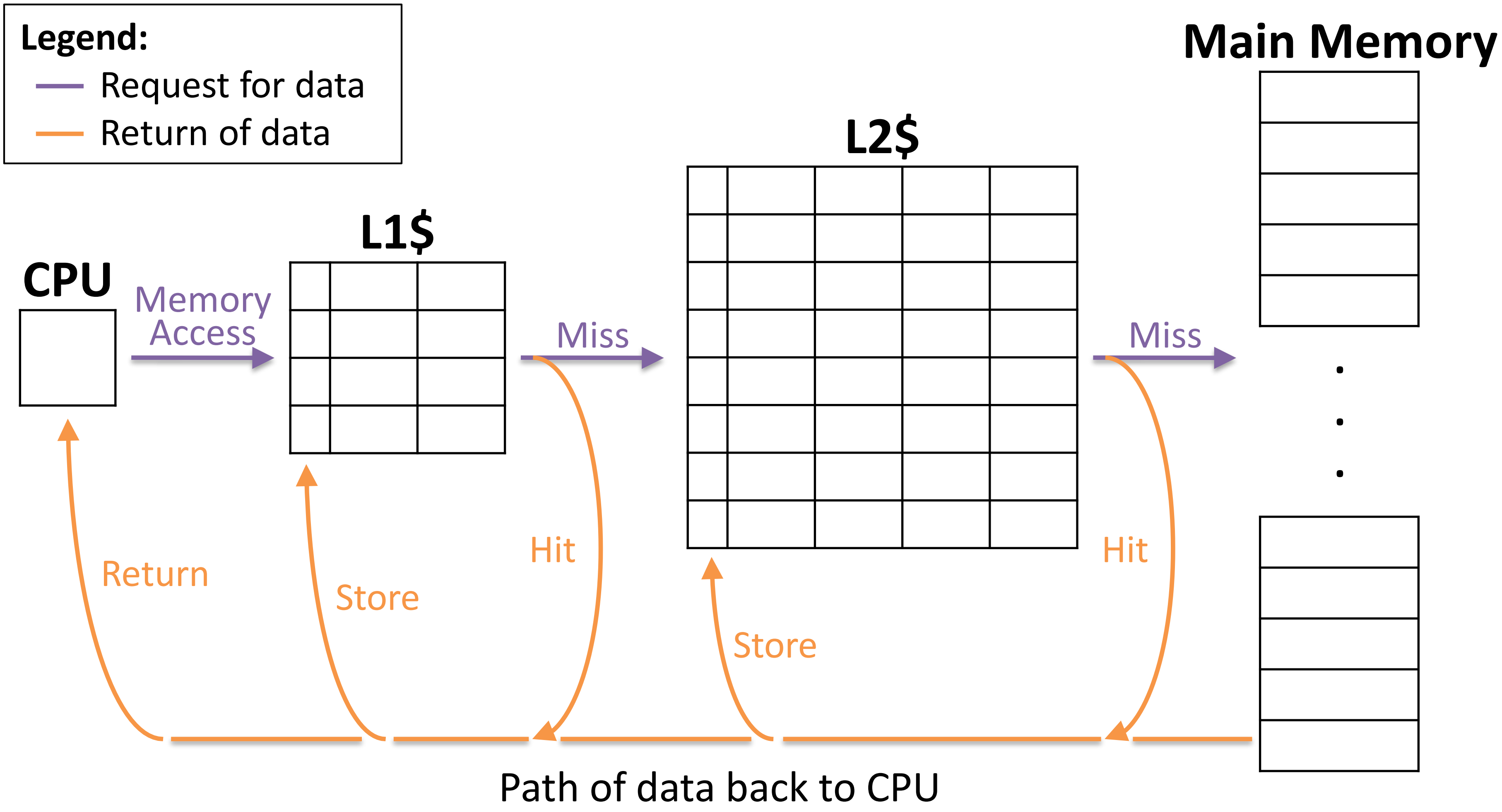
提纲

- 内容主要取材
  - CS61C的11讲和12讲
- 存储层次概述
- 直接映射Cache
- 直接映射Cache举例
- Cache读和写
- Cache性能
- 多级Cache
- 组相连Cache
- 改进Cache性能
- 多级Cache性能实战
- 当代Cache举例

Multiple Cache Levels

- With advancing technology, have more room on die for bigger L1 caches and for L2 (and in some cases even L3) cache
  - Normally lower-level caches are *unified* (i.e. holds both instructions and data)
- Multilevel caching is a way to reduce miss penalty
- So what does this look like?

Multilevel Cache Diagram



Legend:
- 🟪 Request for data
- 🟧 Return of data

Multilevel Cache AMAT

- AMAT = L1 HT + L1 MR × L1 MP
  - Now L1 MP depends on other cache levels
- L1 MP = L2 HT + L2 MR × L2 MP
  - If more levels, then continue this chain (i.e. $MP_i = HT_{i+1} + MR_{i+1} \times MP_{i+1}$)
  - Final MP is main memory access time
- For two levels:

  AMAT = L1 HT + L1 MR × (L2 HT + L2 MR × L2 MP)

Multilevel Cache AMAT Example

- **Processor specs:** 1 cycle L1 HT, 2% L1 MR, 5 cycle L2 HT, 5% L2 MR, 100 cycle main memory HT
  - Here assuming unified L1$
- Without L2$:

$$\text{AMAT}_1 = 1 + 0.02 \times 100 = 3$$

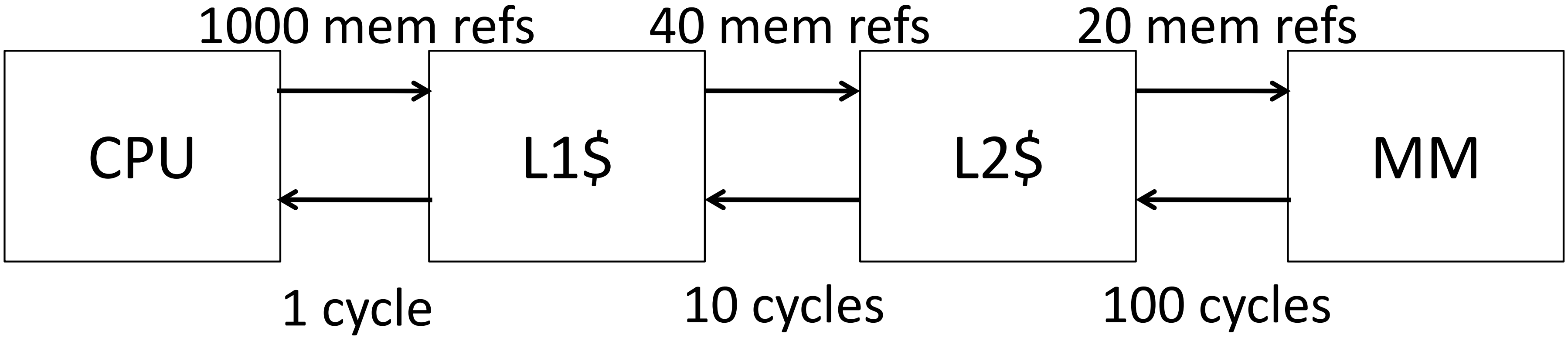
- With L2$:

$$\text{AMAT}_2 = 1 + 0.02 \times (5 + 0.05 \times 100) = 1.2$$

Local vs. Global Miss Rates

- *Local miss rate:* Fraction of references to one level of a cache that miss
  - e.g. L2$ local MR = L2$ misses/L1$ misses
  - Specific to level of caching (as used in AMAT)
- *Global miss rate:* Fraction of all references that miss in all levels of a multilevel cache
  - Property of the overall memory hierarchy
  - Global MR is the product of all local MRs
    - Start at Global MR = L_n misses/L_1 accesses and expand
    - So by definition, *global MR ≤ any local MR*

Memory Hierarchy with Two Cache Levels



- For every 1000 CPU to memory references
  - 40 will miss in L1$; what is the local MR? 0.04
  - 20 will miss in L2$; what is the local MR? 0.5
  - Global miss rate? 0.02

Rewriting Performance

- For a two level cache, we know:

$$\mathrm{MR}_{\mathrm{global}} = \mathrm{L1\ MR} \times \mathrm{L2\ MR}$$

- AMAT:
  - $\mathrm{AMAT} = \mathrm{L1\ HT} + \mathrm{L1\ MR} \times (\mathrm{L2\ HT} + \mathrm{L2\ MR} \times \mathrm{L2\ MP})$

    $= \mathrm{L1\ HT} + \mathrm{L1\ MR} \times \mathrm{L2\ HT} + \color{red}{\mathrm{MR}_{\mathrm{global}}} \times \mathrm{L2\ MP}$
- CPI:
  - $\mathrm{CPI}_{\mathrm{stall}} = \mathrm{CPI}_{\mathrm{base}} + \frac{\mathrm{Accesses}}{\mathrm{Instr}} \times \mathrm{L1\ MR} \times (\mathrm{L1\ MP} + \mathrm{L2\ MR} \times \mathrm{L2\ MP})$
  - $\mathrm{CPI}_{\mathrm{stall}} = \mathrm{CPI}_{\mathrm{base}} + \frac{\mathrm{Accesses}}{\mathrm{Instr}} (\mathrm{L1\ MR} \times \mathrm{L1\ MP} + \color{red}{\mathrm{MR}_{\mathrm{global}}} \times \mathrm{L2\ MP})$

Design Considerations

- L1$ focuses on *low hit time* (fast access)
  - minimize HT to achieve shorter clock cycle
  - L1 MP significantly reduced by presence of L2$, so can be smaller/faster even with higher MR
  - e.g. smaller $ (fewer rows)
- L2$, L3$ focus on *low miss rate*
  - As much as possible avoid reaching to main memory (heavy penalty)
  - e.g. larger $ with larger block sizes (same # rows)

提纲

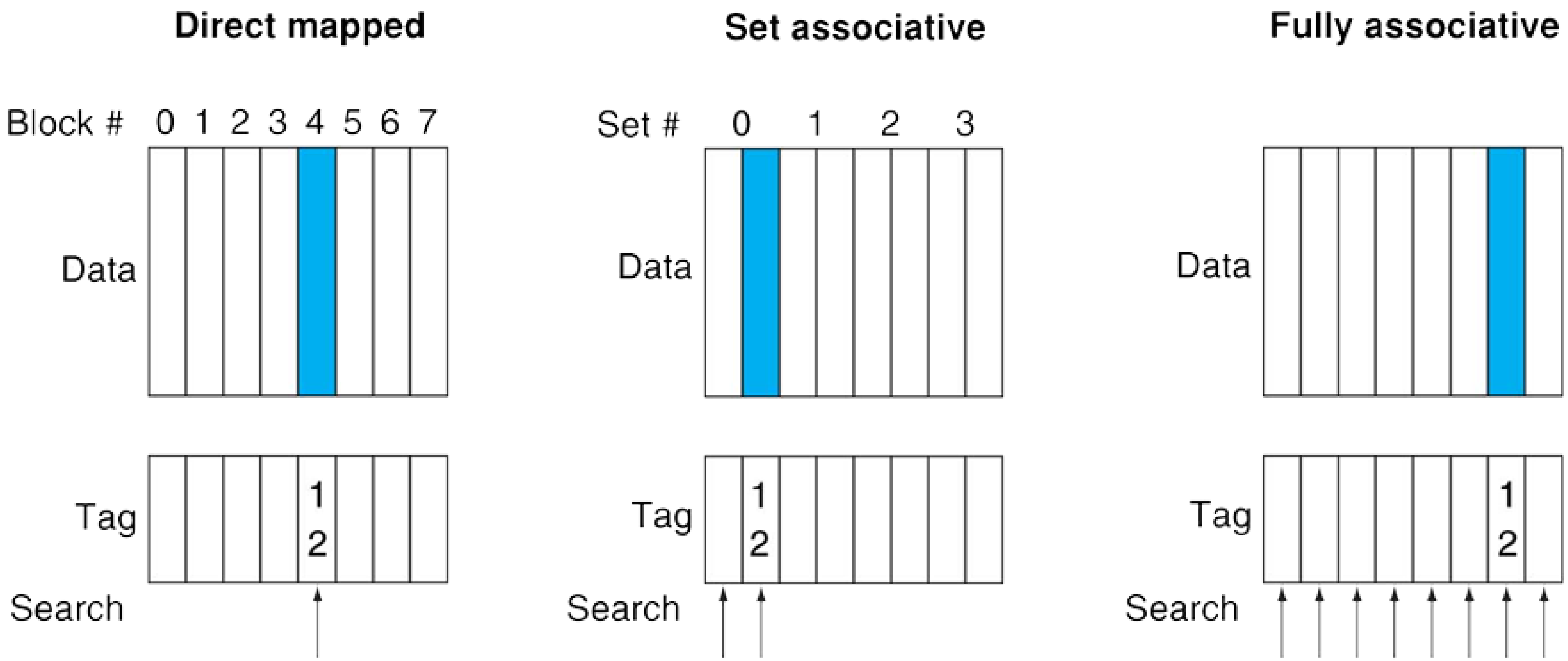
- 内容主要取材
  - CS61C的11讲和12讲
- 存储层次概述
- 直接映射Cache
- 直接映射Cache举例
- Cache读和写
- Cache性能
- 多级Cache
- 组相连Cache
- 改进Cache性能
- 多级Cache性能实战
- 当代Cache举例

Reducing Cache Misses

- Allow more flexible block placement in cache:
- *Direct-mapped:* Memory block maps to exactly one cache block
- *Fully associative:* Memory block can go in any slot
- *N-way set-associative:* Divide $ into sets, each of which consists of n slots to place memory block
  - Memory block maps to a set determined by Index field and is placed in any of the n slots of that set
  - Hash function: (block address) modulo (# *sets* in the cache)

Block Placement Schemes

- Place memory block 12 in a cache that holds 8 blocks



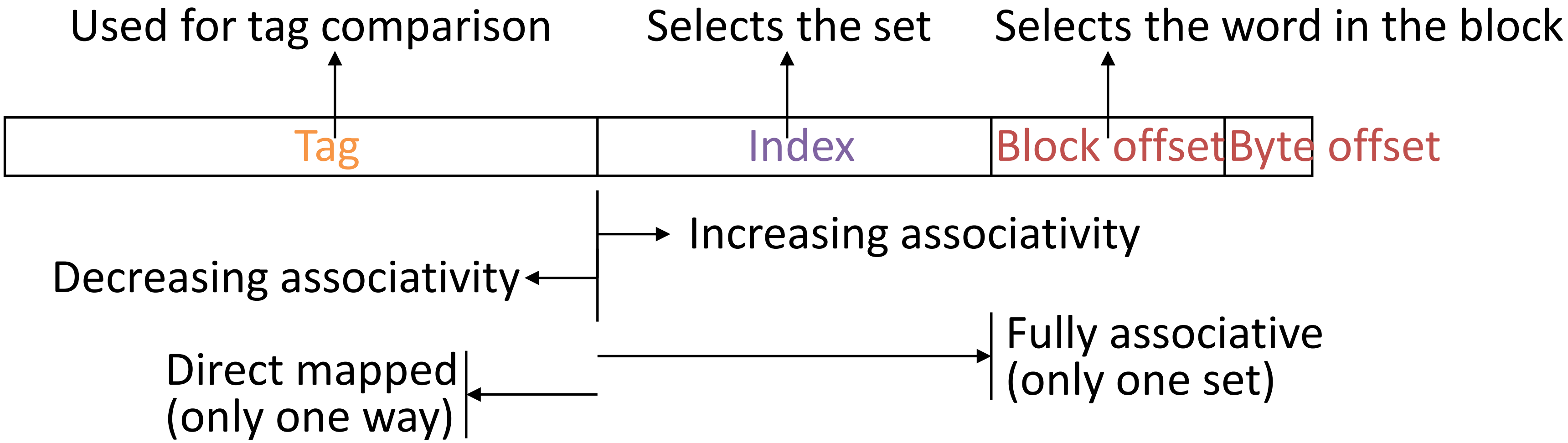
- **Direct-mapped:** Can only go in row (12 mod 8) = 4
- **Fully associative:** Can go in *any* of the slots (1 set/row)
- **2-way set associative:** Can go in either slot of set (12 mod 4) = 0

Effect of Associativity on TIO (1/2)

- Here we assume a cache of fixed size (C)
- **Offset:** # of bytes in a block (same as before)
- **Index:** Instead of pointing to a *row*, now points to a *set*, so I = C/B/associativity
  - Fully associative (1 set): 0 Index bits!
  - Direct-mapped (associativity of 1): max Index bits
  - Set associative: somewhere in-between
- **Tag:** Remaining identifier bits (T = A – I – O)

Effect of Associativity on TIO (2/2)

- For a fixed-size cache, each increase by a factor of two in associativity doubles the number of blocks per set (i.e. the number of slots) and halves the number of sets – decreasing the size of the Index by 1 bit and increases the size of the Tag by 1 bit



Used for tag comparison

Selects the set

Selects the word in the block

Tag

Index

Block offset

Byte offset

Increasing associativity

Decreasing associativity

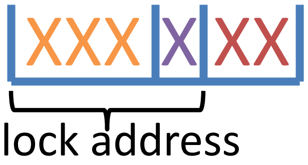
Fully associative
(only one set)

Direct mapped
(only one way)

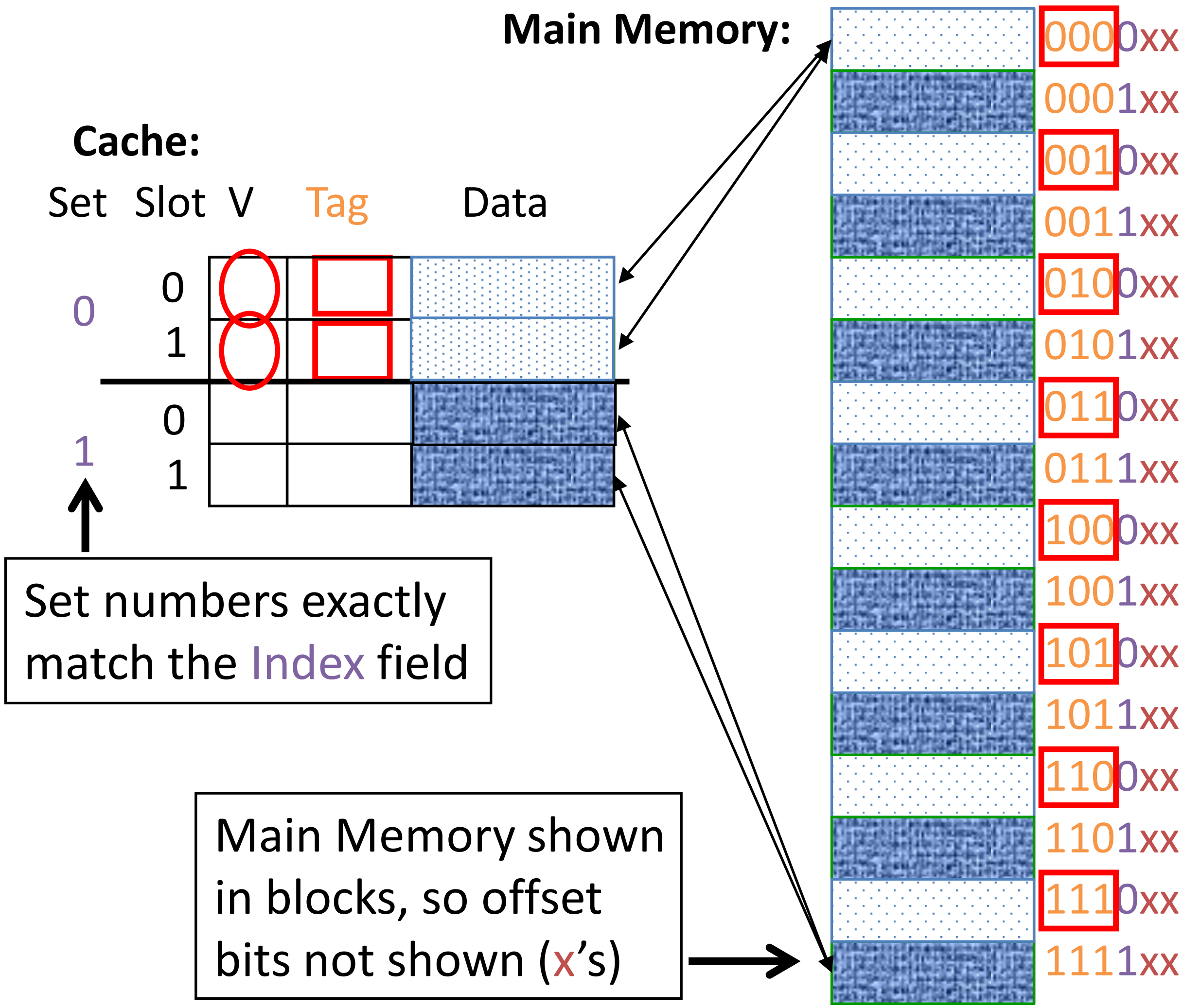
Set Associative Example (1/2)

- Cache parameters:
  - 6-bit addresses, block size of 1 word, cache size of 4 words, 2-way set associative
- How many sets?
  - C/B/associativity = 4/1/2 = 2 sets
- TIO Breakdown:
  - O = $\log_2(4)$ = 2, I = $\log_2(2)$ = 1, T = 6 – 1 – 2 = 3

Memory Addresses:



Set Associative Example (2/2)



Set numbers exactly match the Index field

Main Memory shown in blocks, so offset bits not shown (x’s)

Each block maps into one set (either slot)
(see colors)

On a memory request:
(let’s say 001011$_{\text{two}}$)

1) Take Index field (0)

2) For EACH slot in set, check valid bit, then compare Tag

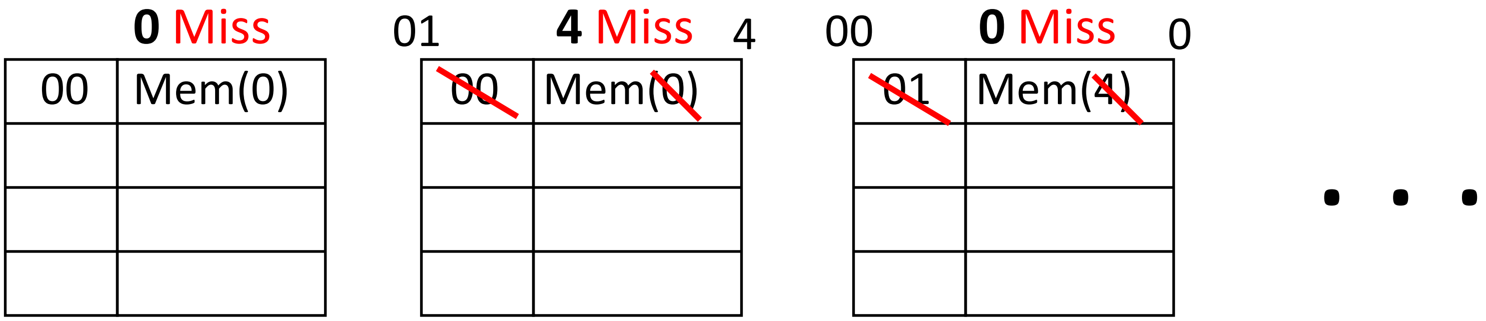
Question: What is the TIO breakdown for the following cache?

- 32-bit address space
- 32 KiB 4-way set associative cache
- 8 words/block

	T	I	O
□	21	8	3
[✓ □]	19	8	5
□	19	10	3
□	17	10	5

Worst-Case for Direct-Mapped

- Example: direct-mapped $ that holds 4 blocks
  - Starts empty (all initially not valid)
- Consider the memory accesses: 0, 4, 0, 4, ...



0 Miss

00	Mem(0)

01 **4** Miss 4

~~00~~	Mem(~~0~~)

00 **0** Miss 0

~~01~~	Mem(~~4~~)

. . .

- HR of 0%
  - Ping pong effect: alternating requests that map into the same cache row

Set Associative Benefits

- Example: 2-way associative $ holds 4 blocks
  - – Starts empty (all initially not valid)
- Consider the memory accesses: 0, 4, 0, 4, ...

0 Miss

000	Mem(0)

4 Miss

000	Mem(0)
010	Mem(4)

0 Hit

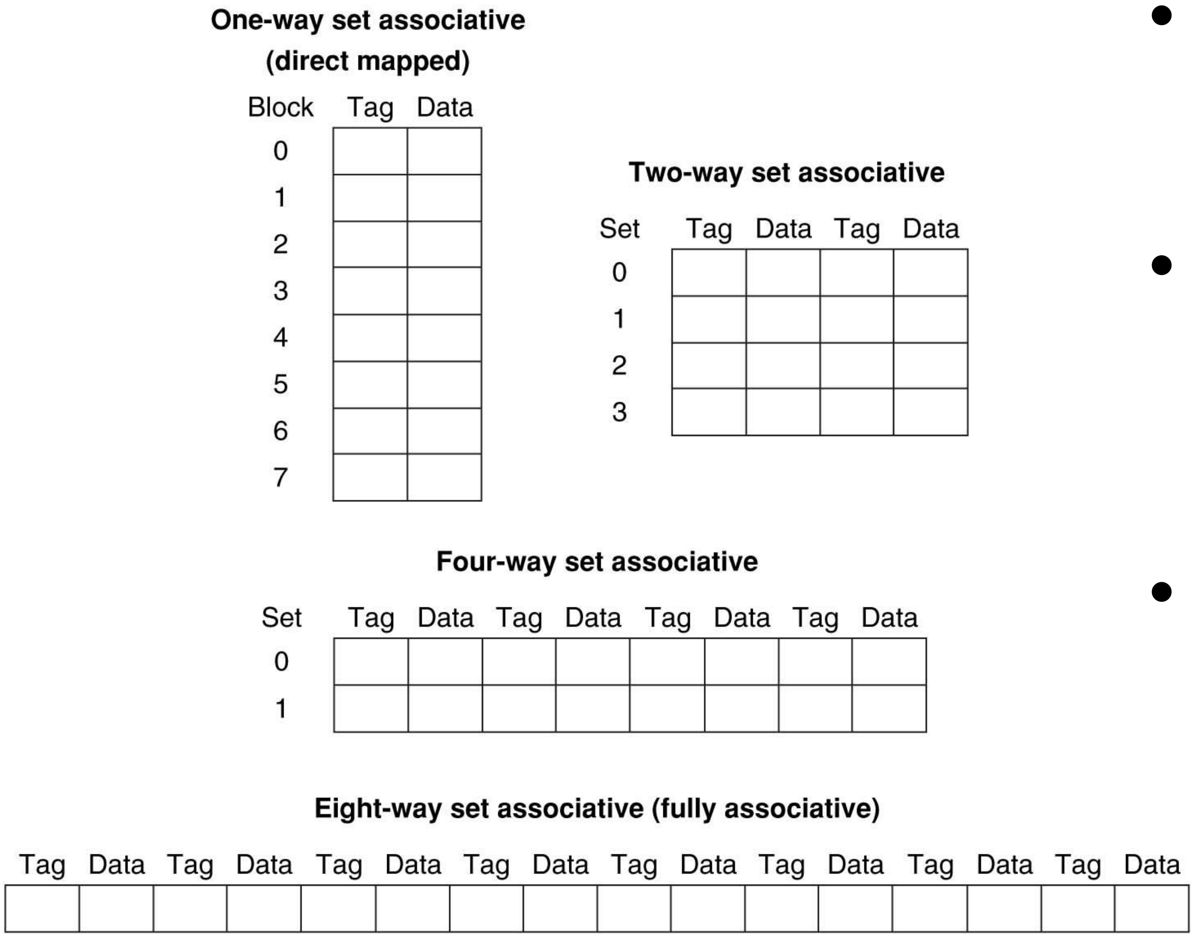
000	Mem(0)
010	Mem(4)

4 Hit

000	Mem(0)
010	Mem(4)

- HR of (n-2)/n – big improvement!
  - – Reduced conflict misses because memory locations that map into the same set can co-exist

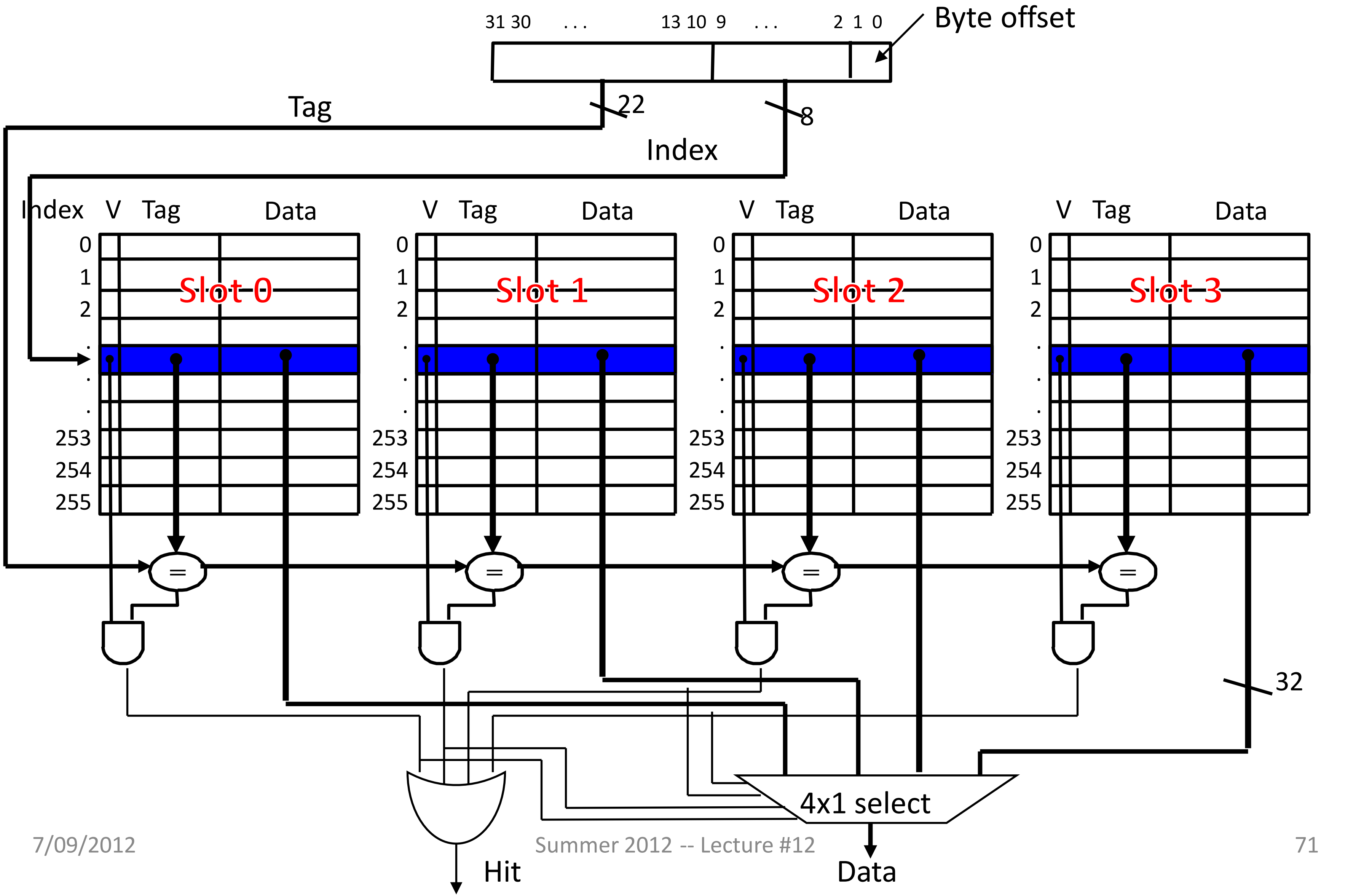
Example: Eight-Block Cache Configs



- Total size of $ = *# sets × associativity*
- For fixed $ size, associativity ↑ means # sets ↓ and slots per set ↑
- With 8 blocks, an 8-way set associative $ is same as a fully associative $

4-Way Set Associative Cache

- 2^8 = 256 sets each with four slots for blocks which size of 1 word



Costs of Set-Associative Caches

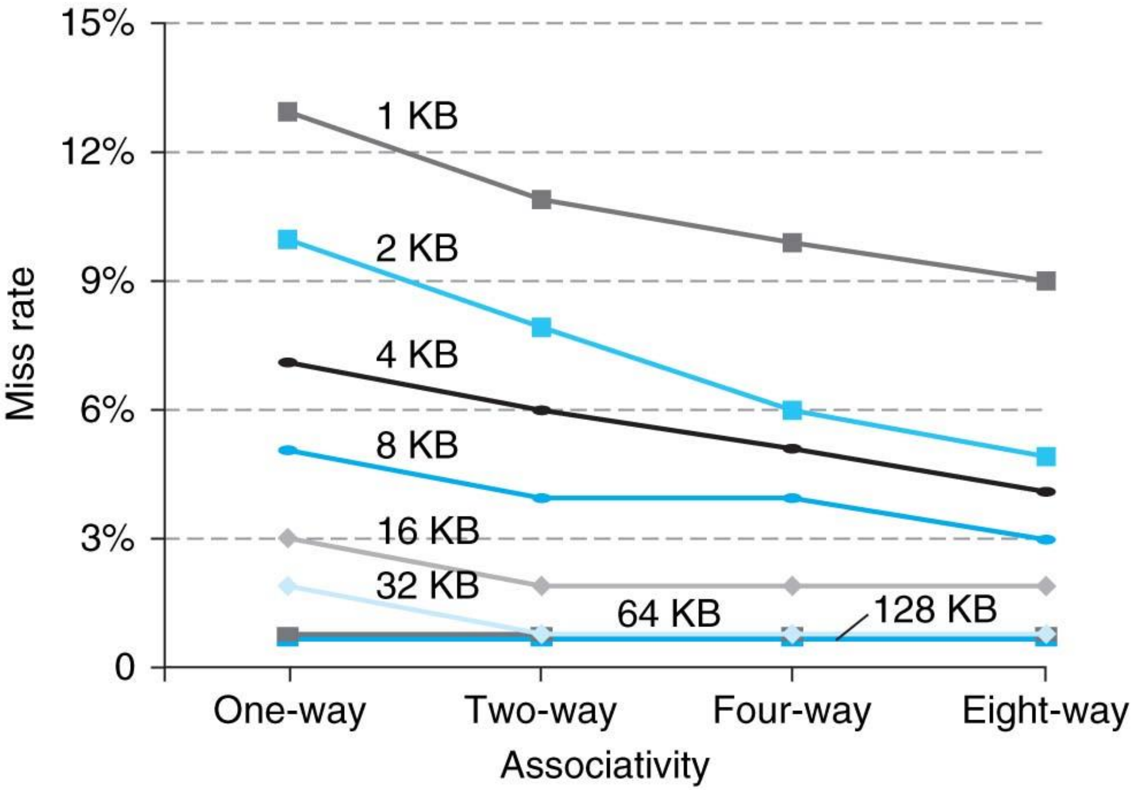
- For *n*-way set associative cache:
  - Need *n* comparators for Tag comparisons
  - Must choose data from correct slot (multiplexer)
- On cache miss, which block do you replace in the set?
  - Use a *cache block replacement policy*
  - There are many (most are intuitively named), but we will just cover a few in this class

http://en.wikipedia.org/wiki/Cache_algorithms#Examples

Block Replacement Policies (1/2)

- Random Replacement:
  - Hardware randomly selects a cache block in set
- Least Recently Used (LRU):
  - Hardware keeps track of access history and replaces the entry that has not been used for the longest time
  - For 2-way set associative cache, can get away with just one bit per set
- Example of a Simple "Pseudo" LRU Implementation:
  - For each set, store a hardware replacement pointer that points to one slot
    - Takes just $\log_2$(associativity) bits
  - Whenever that slot is accessed, move pointer to next slot
    - That slot is the *most* recently used and cannot be the LRU

Benefits of Set-Associative Caches



- Consider cost of a miss vs. cost of implementation
- Largest gains are in going from direct mapped to 2-way (20%+ reduction in miss rate)

提纲

- 内容主要取材
  - CS61C的11讲和12讲
- 存储层次概述
- 直接映射Cache
- 直接映射Cache举例
- Cache读和写
- Cache性能
- 多级Cache
- 组相连Cache
- 改进Cache性能
- 多级Cache性能实战
- 当代Cache举例

Improving Cache Performance (1/2)

1) Reduce the Hit Time of the cache
   - Smaller cache (less to search/check)
   - 1 word blocks (no MUX/selector to pick word)

2) Reduce the Miss Rate
   - Bigger cache (capacity)
   - Larger blocks (compulsory & spatial locality)
   - Increased associativity (conflict)

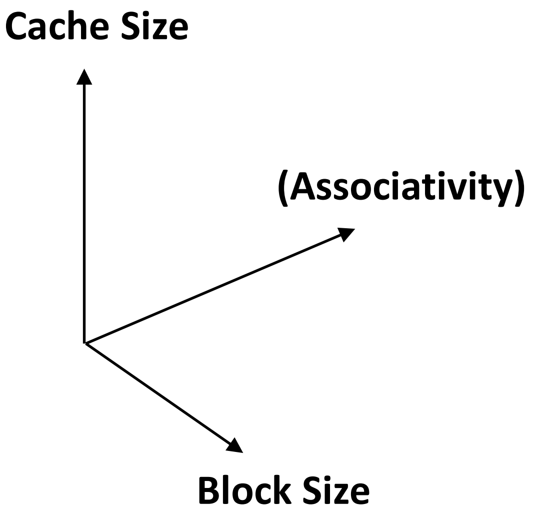
Improving Cache Performance (2/2)

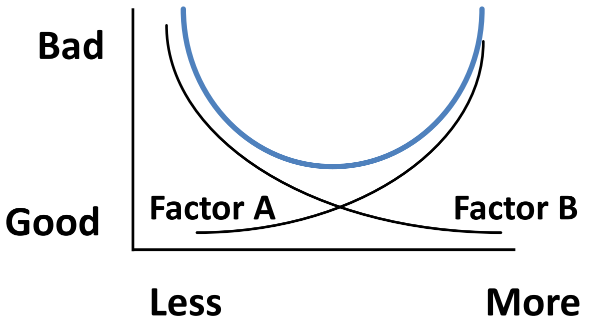
3) Reduce the Miss Penalty
   - Smaller blocks (less to move)
   - Use multiple cache levels
   - Use a *write buffer*
     - Can also check on read miss (may get lucky)

The Cache Design Space

Several interacting dimensions

- Cache parameters:
  - Cache size, Block size, Associativity
- Policy choices:
  - Write-through vs. write-back
  - Write allocation vs. no-write allocation
  - Replacement policy
- Optimal choice is a compromise
  - Depends on access characteristics
    - Workload and use (I$, D$)
  - Depends on technology / cost
- Simplicity often wins





Summary (1/2)

- Cache Performance
  - AMAT = HT + MR × MP
  - CPU time = IC × $\text{CPI}_{\text{stall}}$ × CC
    = IC × (CPI_{base} + Memory-stall cycles) × CC
- Multilevel caches reduce *miss penalty*
  - Local vs. global miss rate
  - Optimize first level to be fast (low HT)
  - Optimize lower levels to not miss (minimize MP)

Summary (2/2)

- Set associativity reduces *miss rate*
  - N-way: cache split into sets, each of which have *n* slots to place memory blocks
  - Fully associative: blocks can go anywhere
  - Memory block maps into more than one slot, reducing conflict misses
- Cache performance depends heavily on cache design (there are many choices)
  - Effects of parameters and policies
  - Cost vs. effectiveness

提纲

- 内容主要取材
  - CS61C的11讲和12讲
- 存储层次概述
- 直接映射Cache
- 直接映射Cache举例
- Cache读和写
- Cache性能
- 多级Cache
- 组相连Cache
- 改进Cache性能
- 多级Cache性能实战
- 当代Cache举例

Multilevel Cache Practice (1/3)

- **Processor specs:**
  - CPI_{base} of 2
  - 100 cycle miss penalty to main memory
  - 25 cycle miss penalty to unified L2$
  - 36% of instructions are load/stores
  - 2% L1 I$ miss rate; 4% L1 D$ miss rate
  - 0.5% **global** U(nified)L2$ miss rate
- What is $\text{CPI}_{\text{stall}}$ with and without the L2$?

Multilevel Cache Practice (2/3)

- **Notes:**
  - Both L1 I$ and L1 D$ send misses to L2$
  - What does the global L2$ MR mean?
    - MR to main memory
    - Since there are 2 L1$s, implies L2$ has 2 different local MRs depending on source of access
  - Use formula shown at bottom of slide 58

Multilevel Cache Practice (3/3)

- **Without L2$:**

$$CPI_{stall} = 2 + \overset{\text{Instr Fetch}}{1\times0.02\times100} + \overset{\text{Load/Store}}{0.36\times0.04\times100}$$

$$CPI_{stall} = 5.44$$

- **With L2$:**

$$CPI_{stall} = 2 + \boxed{1x.02\times25 \quad + .36\times.04\times25}\ \text{L1}$$

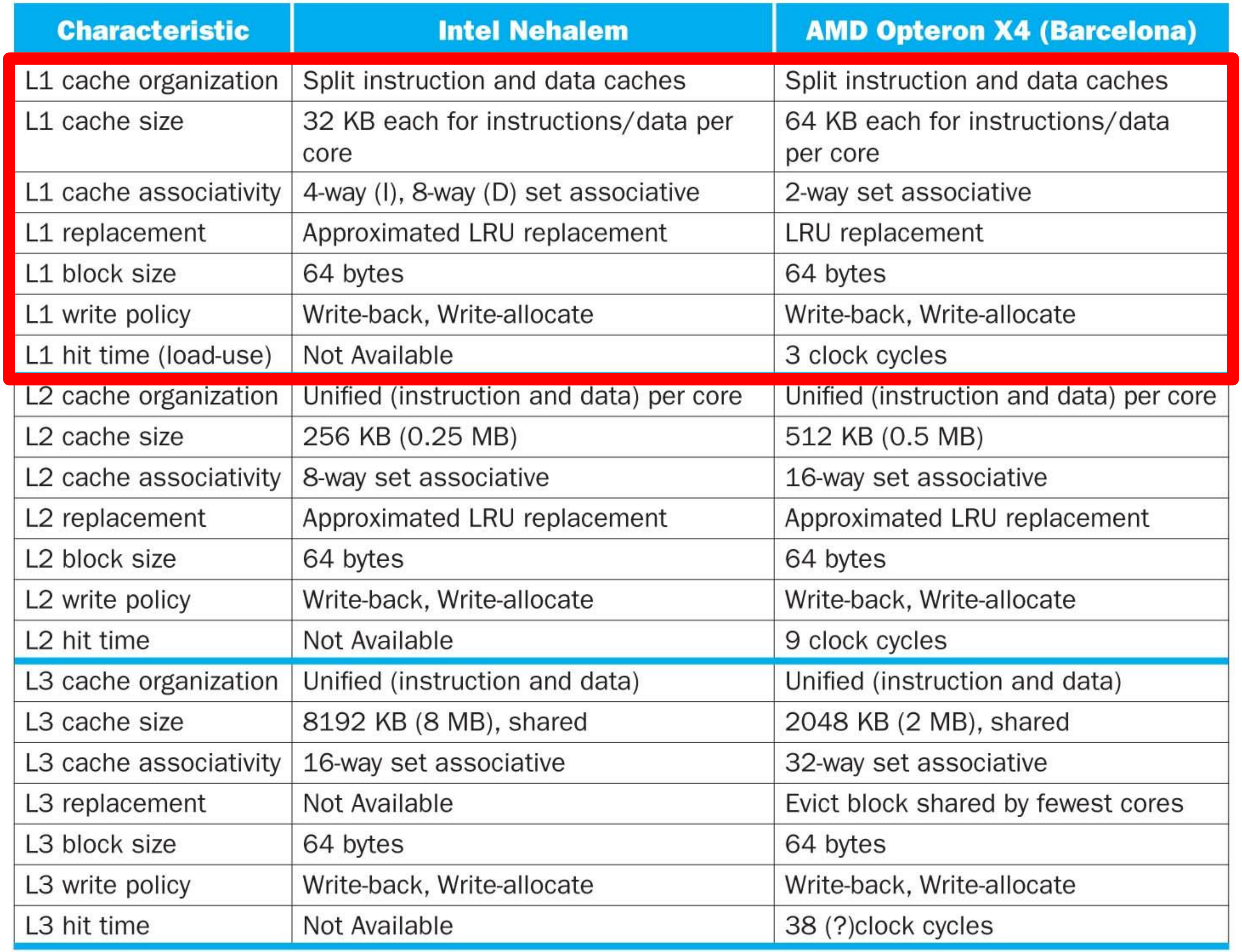
$$+ \boxed{1x.005\times100 + .36\times.005\times100}\ \text{L2}$$

$$= 3.54$$

- CPI_{base} of 2
- 100 cycle miss penalty to main memory
- 25 cycle miss penalty to unified L2$
- 36% of instructions are load/stores
- 2% L1 I$ miss rate; 4% L1 D$ miss rate
- 0.5% **global** U(nified)L2$ miss rate

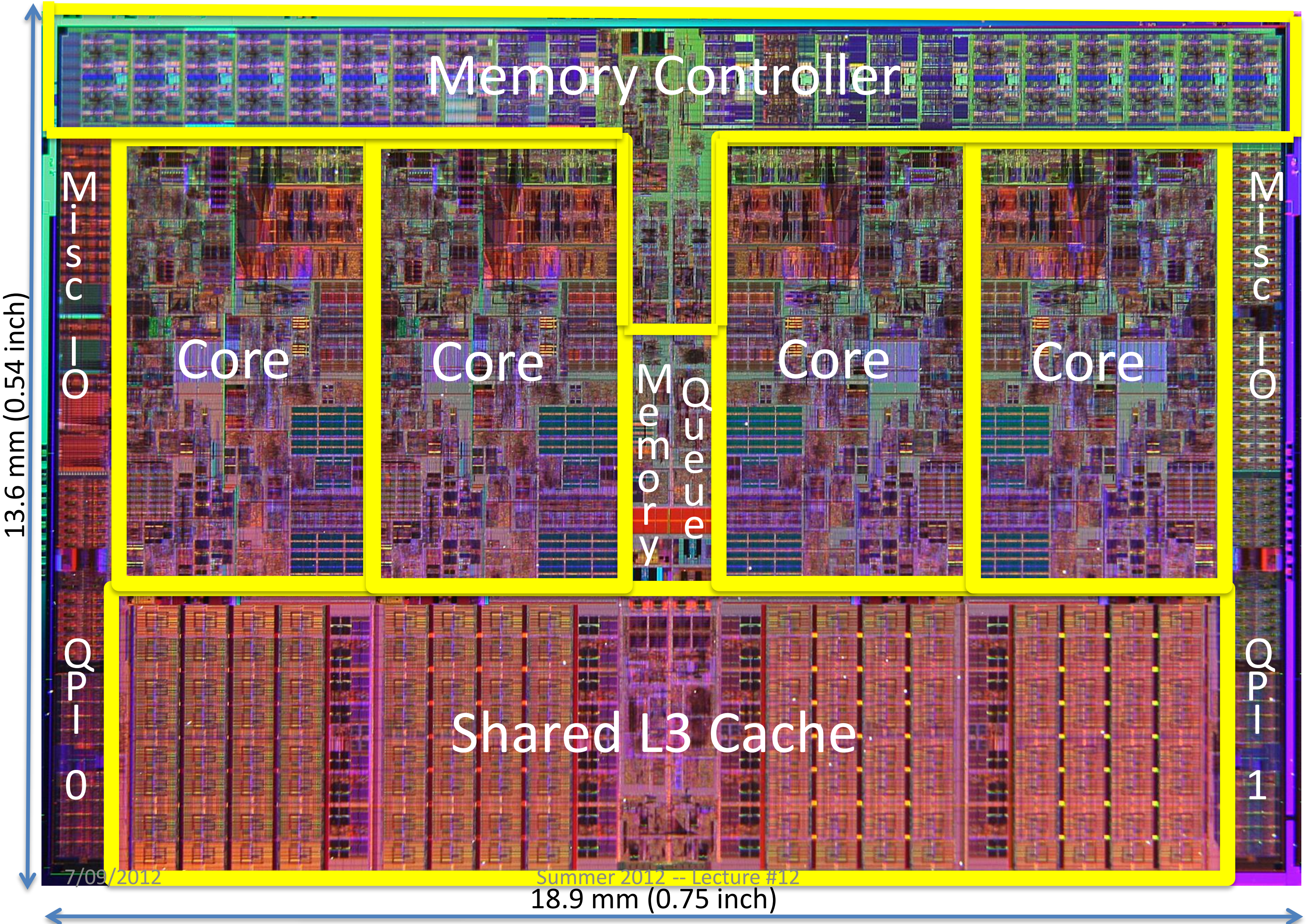
提纲

- 内容主要取材
  - CS61C的11讲和12讲
- 存储层次概述
- 直接映射Cache
- 直接映射Cache举例
- Cache读和写
- Cache性能
- 多级Cache
- 组相连Cache
- 改进Cache性能
- 多级Cache性能实战
- 当代Cache举例



Characteristic	Intel Nehalem	AMD Opteron X4 (Barcelona)
L1 cache organization	Split instruction and data caches	Split instruction and data caches
L1 cache size	32 KB each for instructions/data per core	64 KB each for instructions/data per core
L1 cache associativity	4-way (I), 8-way (D) set associative	2-way set associative
L1 replacement	Approximated LRU replacement	LRU replacement
L1 block size	64 bytes	64 bytes
L1 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L1 hit time (load-use)	Not Available	3 clock cycles
L2 cache organization	Unified (instruction and data) per core	Unified (instruction and data) per core
L2 cache size	256 KB (0.25 MB)	512 KB (0.5 MB)
L2 cache associativity	8-way set associative	16-way set associative
L2 replacement	Approximated LRU replacement	Approximated LRU replacement
L2 block size	64 bytes	64 bytes
L2 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L2 hit time	Not Available	9 clock cycles
L3 cache organization	Unified (instruction and data)	Unified (instruction and data)
L3 cache size	8192 KB (8 MB), shared	2048 KB (2 MB), shared
L3 cache associativity	16-way set associative	32-way set associative
L3 replacement	Not Available	Evict block shared by fewest cores
L3 block size	64 bytes	64 bytes
L3 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L3 hit time	Not Available	38 (?)clock cycles

Intel Nehalem Die Photo



Core Area Breakdown

32KiB I$ per core

32KiB D$ per core

512KiB L2$ per core

Share one 8-MiB L3$